POLITECHNIKA KRAKOWSKA

IM. TADEUSZA KOŚCIUSZKI

WYDZIAŁ FIZYKI MATEMATYKI I INFORMATYKI

KIERUNEK INFORMATYKA

ROBERTO JAVIER SOTO ENTRENA

FACE DETECTION IN C#

PRACA DYPLOMOWA

Promotor: Pawel Jarosz, PhD

Kraków 2012

RESUMEN

Actualmente la detección del rostro humano es un tema difícil debido a varios parámetros implicados. Llega a ser de interés cada vez mayor en diversos campos de aplicaciones como en la identificación personal, la interface hombre-máquina, etc. La mayoría de las imágenes del rostro contienen un fondo que se debe eliminar/discriminar para poder así detectar el rostro humano.

Así, este proyecto trata el diseño y la implementación de un sistema de detección facial humana, como el primer paso en el proceso, dejando abierto el camino, para en un posible futuro, ampliar este proyecto al siguiente paso, que sería, el Reconocimiento Facial, tema que no trataremos aquí.

En la literatura científica, uno de los trabajos más importantes de detección de rostros en tiempo real es el algoritmo de *Viola and Jones*, que ha sido tras su uso y con las librerías de Open CV, el algoritmo elegido para el desarrollo de este proyecto.

A continuación explicaré un breve resumen sobre el funcionamiento de mi aplicación.

Mi aplicación puede capturar video en tiempo real y reconocer el rostro que la Webcam captura frente al resto de objetos que se pueden visualizar a través de ella. Para saber que el rostro es detectado, éste es recuadrado en su totalidad y seguido si este mueve. A su vez, si el usuario lo desea, puede guardar la imagen que la cámara esté mostrando, pudiéndola almacenar en cualquier directorio del PC.

Además, incluí la opción de poder detectar el rostro humano sobre una imagen fija, cualquiera que tengamos guardada en nuestro PC, siendo mostradas el número de caras detectadas y pudiendo visualizarlas sucesivamente cuantas veces queramos.

Para todo ello como bien he mencionado antes, el algoritmo usado para la detección facial es el de *Viola and Jones*. Este algoritmo se basa en el escaneo de toda la superficie de la imagen en busca del rostro humano, para ello, primero la imagen se transforma a escala de grises y luego se analiza dicha imagen, mostrando como resultado el rostro encuadrado.

ABSTRACT

Currently the detection of human face is a difficult issue due to various parameters involved. Becomes of increasing interest in various fields of applications such as personal identification, the man-machine interface, etc. Most of the face images contain a fund to be removed / discriminate in order to detect the human face.

Thus, this project is the design and implementation of a human face detection system, as the first step in the process, leaving the way open for a possible future, extend this project to the next step would be, Facial Recognition , a topic not covered here.

In the literature, one of the most important face detection in real time is the algorithm of Viola and Jones, who has been after use with Open CV libraries, the algorithm chosen for the development of this project.

I will explain a brief summary of the performance of my application.

My application can capture video in real time and recognize the face that the Webcam Capture compared to other objects that can be viewed through it. To know that the face is detected, it is fully boxed and followed if this move. In turn, if the user may want to save the image that the camera is showing, could store in any directory on your PC.

I also included the option to detect the human face on a still image, whatever we have stored in your PC, being shown the number of faces detected and can view them on more times.

For all as well I mentioned before, the algorithm used for face detection is that of Viola and Jones. This algorithm is based on scanning the entire surface of the image for the human face, for this, first the image is converted to gray-scale and then analyzed the image, showing results in the face framed.

PROYECTO FIN DE CARRERA

TEMA: Uso lenguaje C# para la detección facial humana

AUTOR: Roberto Javier Soto Entrena

TUTOR: Dr Inż. Paweł Jarosz

DEPARTAMENTO: Institute of Computer Science

CENTRO DE LECTURA: Faculty of Physics, Mathematics and Computer Science

Fecha de Lectura: 2 – Julio - 2012

Calificación: (A) ECTS

RESUMEN DEL PROYECTO:

El principal objetivo de mi proyecto es que a través de un entorno u aplicación, con la ayuda de nuestra cámara web del ordenador, ésta pueda reconocer y diferenciar un rostro humano de entre otros objetos, que la cámara pueda captar. Para ello usaré lenguaje C# con las correspondientes librerías necesarias.

FINAL PROJECT

TEMA: Using C# language for face detection

AUTOR: Roberto Javier Soto Entrena

TUTOR: Dr Inż. Paweł Jarosz

DEPARTAMENTO: Institute of Computer Science

CENTRO DE LECTURA: Faculty of Physics, Mathematics and Computer Science

Fecha de Lectura: 2 - July - 2012

Calificación: (A) ECTS

RESUMEN DEL PROYECTO:

The main goal of my Final Project is that through an environment or application, with the help of our computer web cam, it could recognize and distinguish a human face from other objects that the camera could capture. To do this I will use C# language with the appropriate libraries necessary.

ACKNOWLEDGEMENT

To my Family, Pedro, Pilar, Luis, Bárbara, Champi (Mormol) y Corbata (Mi Negra)

Because they were and are always there, no matter what happen and because of they, now I am here, THANK YOU

To my Uncle, Jorge and my Aunt, Maria Isabel

Just giving me love and support during all this years

To Nestor my Friend

Because without his support and true friendship, I will not managed to get here

To my Friends, Javi and Guille

Without their support outside of the University it would have been hardest

To my girlfriend, Agata

Especially for the last months, her helps and supports. Thank you.

To my Polish Teacher, Dr. Inz. Pawel Jarosz

Because he supports me and tried to help and represent me in the hardest moments of my studies.

To all the close people that I know that they were there just

Thanks

RESUMEN

El principal objetivo de mi proyecto es que a través de un entorno u aplicación, con la ayuda de nuestra cámara web del ordenador, ésta pueda reconocer y diferenciar un rostro humano de entre otros objetos, que la cámara pueda captar. Para ello usaré lenguaje C# con las correspondientes librerías necesarias.

El primer paso de todo fue aprender el uso del nuevo lenguaje de programación C# y buscar las herramientas necesarias para poder desarrollar esta aplicación. Para ello se han usado:

- Visual studio 2010 Professional
- Librerías OpenCV
- Equipo Portátil
- Cámara Web (integrada en el equipo portátil)

El aprendizaje del lenguaje C#, programación orientada a objetos, corrió por mi parte, usando bibliografía proporcionada por mi tutor y como no, por la proporcionada por internet y su amplia oferta de tutoriales.

El motivo por el cuál se usó C#, es por su potencia y posible ampliación en un futuro con el Reconocimiento facial, frente a mi idea inicial de desarrollarlo en MatLab.

Planificación por etapas

El fin y principal reto de este proyecto era la detección facial humana en vivo, es decir, mediante la captura continua por parte de la cámara, de tal manera que se pudiera identificar la cara del sujeto si este se moviese.

Tras la superación de los determinados hitos, se propuso la inclusión del mismo sistema de detección facial pero aplicada a una imagen fija.

Poco a poco se le fueron añadiendo mejoras tales como identificador numérico de las caras reconocidas en la imagen fija y su almacenamiento y muestra en secuencia de sólo la parte detectada, es decir, el rostro.

Por todo ello la aplicación final, dispone de un entorno en el cual dentro de la misma ventana se incluyen todos estos elementos.

Creación de la aplicación

A continuación explicaré de manera resumida los pasos que he seguido.

Lo primero de todo y tras una larga etapa de documentación y aprendizaje, opte por el uso de las librerías EmguCV, que sirve de puente para las librerías de OpenCV (**Open** Source **Computer Vision**). Para poder usarlas hay que instalarlas en el ordenador y tras esto añadir las referencias necesaria a Visual Studio 2010 para su posterior uso.

En concreto se instaló la versión:

Emgucv2.2.1.1150

Tras este paso, lo primero que se debe comprobar es el funcionamiento de la cámara web integrada, para ello se creó una simple ventana donde se podía visualizar lo que la cámara capta.

Después se añadió la opción de poder elegir múltiples cámaras, es decir, que el programa primero ve si hay cámaras instaladas y si las hay, se usan. O también se ha dejado abierto el caso en el que se tengan varias cámaras y el usuario pueda seleccionar la que él quiera.

Ahora es el momento de comenzar a diseñar código para la inclusión de las librerías necesarias para poder detectar un rostro humano.

Gracias a las librerías OpenCV / EmguCv, usaremos la función Haar Cascade, la cual se trata de un clasificador (detector) probado en miles de rostros humanos. Este clasificador usa el algoritmo de Viola and Jones, del que hablaremos más adelante.

Gracias a esta función Haar, podemos calibrar como nuestro programa escaneará la imagen, y según cambiemos sus parámetro obtendremos resultados más toscos o más finos, es decir, se detectaran más o menos rostros, pero con la consiguiente carga o disminución del tiempo que ello conlleva. Esta es la función a usar:

```
var faces = grayframe.DetectHaarCascade (haar, 1.4, 4,
HAAR_DETECTION_TYPE.DO_CANNY_PRUNING, new Size (25, 25)) [0];
```

Para terminar, como bien se comentó al principio, se le añaden una serie de extras como contador de número de caras detectadas en la imagen además de mostrar en otra ventana dicho rostro o rostros en secuencia, dependiendo de cuantos se hayan detectado.

Algoritmos

Bien, ahora entremos más en profundidad, sobre los algoritmos y funciones que hacen que todo esto sea posible.

Comencemos plateando los problemas que nos encontraremos:

- Teniendo nuestra imagen, es identificar sobre toda la región de esta imagen que es un rostro y que no lo es.
- Este problema es un desafío debido a los siguiente factores:

    - Condiciones de la imagen
    - Expresión facial
    - Pose
    - Presencia o ausencia de elementos estructurales, ejemplo gafas, etc.
    - Orientación de la imagen

Otro problema relacionado será:

- Rastreo de rostros: estimar continuamente la ubicación de un rostro en una secuencia de imágenes en tiempo real, es decir, video.

Algoritmo de Viola and Jones

Este algoritmo es capaz de procesar imágenes extremadamente rápido alcanzando altos índices de detección.

Para ello usa tipo Haar. Que trata de buscar y determinar características basadas en sumas y restas de los niveles de intensidad de la imagen, el valor de una característica está dado por la diferencia de la suma de todos los pixeles de color negro con las suma de todos los pixeles de color blanco.

El valor de la imagen integral en la posición x, y, contiene la suma de todos los pixeles arriba y a la izquierda de x, y, inclusive.

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

$$ii(x,y) = i(x,y) - ii(x-1,y) - ii(x,y-1) + ii(x-1,y-1)$$

Usando la imagen integral cualquier suma rectangular puede ser calculada con cuatro referencias a memoria.

$$ii(C) - ii(B) - ii(D) + ii(A)$$

Características 2-rectangulares: 6 referencias
Características 3-restangulares: 8 referencias
Características 4-rectangulares: 9 referencias

Por tanto:

- La eficiencia computacional de las características compensa ampliamente su limitada flexibilidad.
- El detector examina la entrada en muchas escalas cada una de 1.25 mayor que la anterior.
- El conjunto de características tienen la propiedad de que pueden ser evaluadas a cualquier escala con unas cuantas operaciones.
- Aunque cada característica puede ser calculada eficientemente, calcular el conjunto completo es excesivamente costoso.
- El algoritmo trabaja bajo la hipótesis de que un número pequeño de características pese combinando para formar un clasificador eficiente.
- Para seleccionar las características y entrenar al clasificador se usa variante de AdaBoost, combinando una selección de funciones débiles de clasificación para formar un clasificador fuerte.
- Los clasificadores deben resolver una serie de problemas de aprendizaje. Luego de cada ronda, los ejemplos son re-ponderados para asignar los que fueron clasificados de forma incorrecta.
- El clasificador fuerte final es una combinación de clasificadores débiles junto con un umbral.
- Lo que se busca es asociar un mayor peso a los buenos clasificadores y un menor peso a los malos clasificadores.

- Para cada característica el clasificador débil determina el umbral óptimo de la función de clasificación, tal que el mínimo número de ejemplos sean clasificados incorrectamente.
- Un clasificador débil $(h_j(x))$ consiste de una característica (f_j), un umbral (θ_j)y una paridad (p_j) indicando la dirección del signo de la desigualdad.
- $h_j(x) = [p_j f_j(x) < p_j \theta_j]$, x es una ventana de 25 x 25 de una imagen.

Con el algoritmo de Boosting conseguimos que:

Dadas las imágenes ejemplo $(x_1, y_1), \dots, (x_n, y_n)$ donde $y_i = 0{,}1$ para los ejemplos negativos y positivos respectivamente.

E inicializa los pesos $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ para $y_i = 0{,}1$ respectivamente, donde m y l son el número de ejemplos positivos y negativos, respectivamente.

Por tanto para $t = 1, \dots., T$: Tenemos:

- Normalizar los pesos: $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$

- Para cada característica j, el error es evaluado con respecto a

$$w_t, e_j = \sum_i \left| h_j(x_i) - y_i \right|$$

- Escoge un clasificador, h_t con el menor error e_t.
- Actualiza los pesos $w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$, donde $e_i = 0$ si el ejemplo x_i es clasificado correctamente, en caso contrario

$$e_i = 1, \text{ y } \beta_t = \frac{e_t}{1-e_t}$$

- El clasificador fuerte final es:

$$h(x) = [\sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t], \text{ donde } \alpha_t = \log \frac{1}{\beta_t}$$

Con todo ellos la Cascada de clasificadores, se construye una cascada de clasificadores la cual alcanza mejor rendimiento en la detección, al mismo tiempo que reduce el tiempo de cálculo y las fases en la cascada son construidas entrenando clasificadores usando AdaBoost.

INDEX

INDEX FIGURE

Chapter 1

Introduction

1.1 Context

The analysis of biometric patrons during last years had a big number of applications like check identity in airports or even face detection in our cameras. So if a field were many people try to develop it and make it better day by day.

For all of this I tried to create a useful and strong application and at the same time easy to use, bad with the best quality that I could.

The first step to make it possible is the face detection in a picture. In the beginning my idea was to make it directly on live video, but as I could see and with the advice of my teacher I started to create my application with a picture.

Ones that we have it, we have to extract the characteristics of the picture using different kind of algorithms that I will explain later. But to make a small resume of what I used is the Viola and Jones Algorithm, which is very powerful, giving extra ordinaries results between to parameters (those parameters we could management to choose what we prefer, high velocity, but less rates or high rates with less velocity), velocity and high rates of detection.

For all of this the pictures or live cam have a background that have to be discriminated for have our face detection.

1.2 Objectives

The project tries the design and implementation of a system of human face detection in language C#.

As I mention before for make it possible the algorithm of Viola and Jones is essential, and to make it possible, is necessary to use the free libraries of OpenCV / EmguCV, because of that we choose the Viola and Jones algorithm.

The main goal in this work is to identify a human face detection method in real time and on static images, take a detailed description of its stages and its implementation based on OpenCV library. All of this allows designing the application for static images or video for any type of webcam.

Chapter 2

Environment project

2.1 Analysis of the requirements

To create this project will take into consideration the following things:

- Conditions of illumination, that can affect to it, for example with to much shadows will be hard the detection.
- It does not matter how the background its, the face must be detected.
- The introductions of extra object to the face could make it hard to identify.

2.2 Hardware used

This project barns looking for a cheap solution, for all of this, are used elements that are easy to find or even are common that normal people have, that mean that is easily access to domestic market with low price. The elements hardware used are:

- Personal Laptop (it can be Personal Computer).
- Minimum Requirements to run the Software used:
    - CPU of 1,6Hz or higher
    - 1024 MB RAM
    - 3 GB Hard Disk
    - Velocity of 5400 RPM
    - Video Card allows DirectX 9 and minimum resolution 1280 x 1024
    - DVD - ROM
- Integrated Webcam (also it can be used external Webcam with no problem of compatibilities).

2.3 Software used

To could program all this application in C# is necessary an integrated development environment. Is used:

- Visual Studio 2010 (Microsoft License).
- Libraries of OpenCV / EmguCV (needed for could make human face detection).
- In my case Window 7 Home Edition.

2.4 Stage

The standard procedure of the system functionality cons of the steps showed in the figure 2.1 and in the successive sections will be explained deeper each one of it.

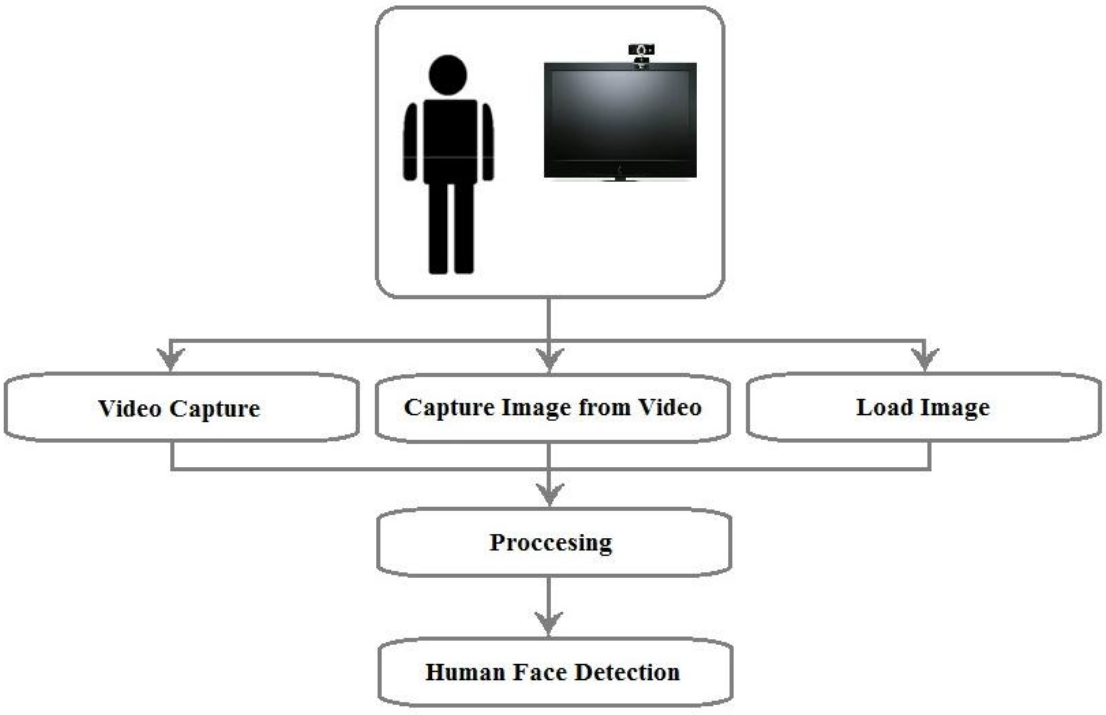

Figure 1 Scheme of Application

2.4.1 Capture Image

The OpenCV library has an interface that provides and abstraction layer to obtain the images in an easy way from a Webcam. This abstraction layer allows us capture it just knowing the Identifier of our device.

In this step we can a group the three ways to obtain what we want, **Video Capture, Capture Image from Video, Load Image.**

An example of the working camera stage, one picture is from **Video Capture** (Figure 2.2) and the other, the same PictureBox, is used for **Capture Image from Video** and **Load Image** (Figure 2.3):

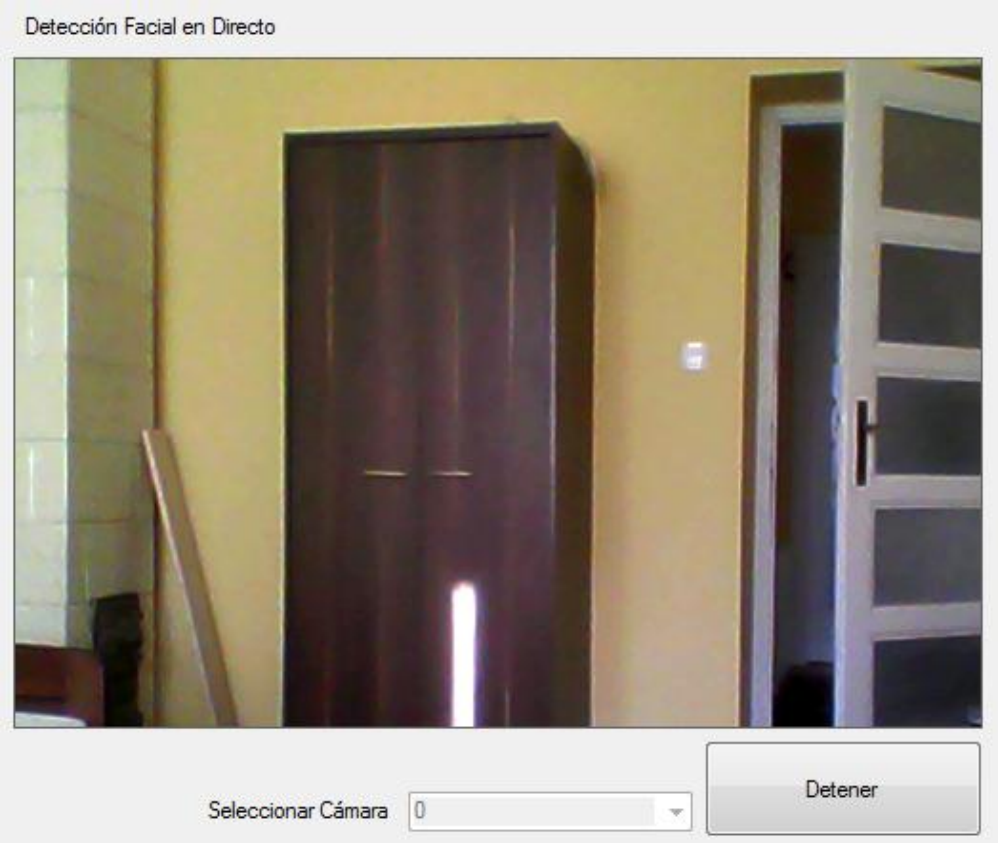

Figure 2 Live Capture with the Webcam

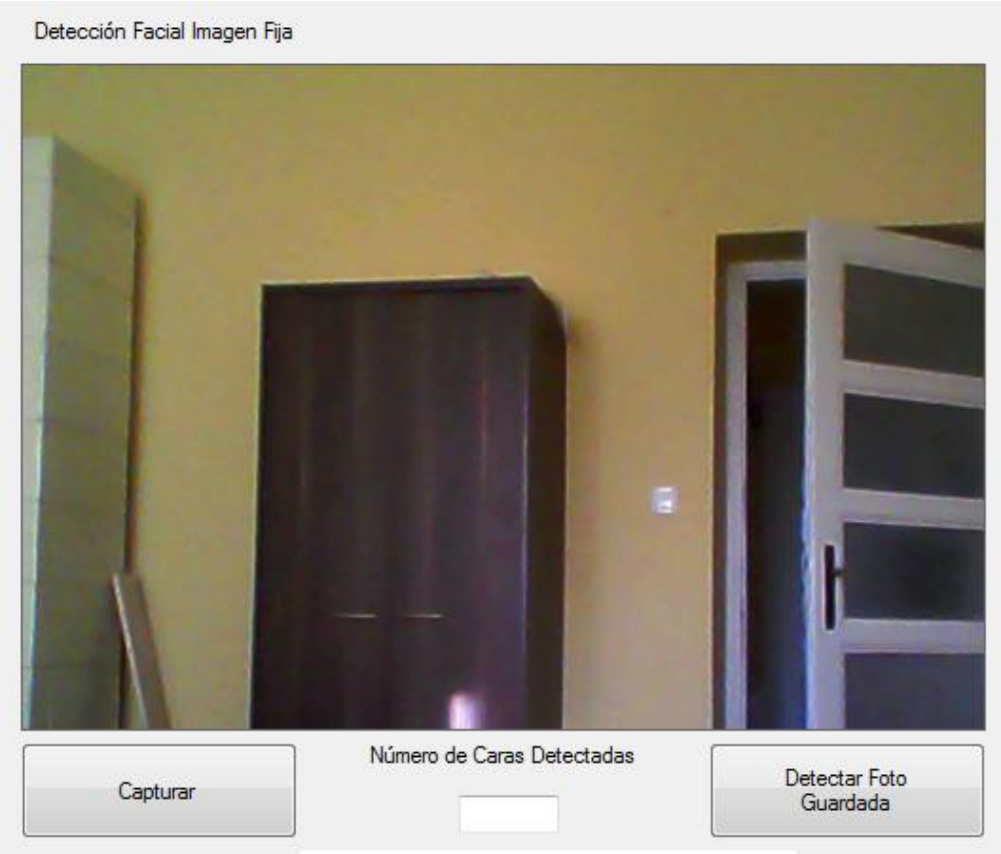

Figure 3 Capture Image with the Webcam

To could make all of these, first we have to download the library from:

http://sourceforge.net/projects/emgucv/files/emgucv/

The version used is:

Emgucv2.2.1.1150

Then we install it and add the reference to Visual Studio 2010 to could use it.

2.4.2 Processing and Human Face Detection

I will lump together this to steps because they are so nearly and one without other, nothing could be work.

First of all, we have to understand why we are using OpenCV and what base is of. To understand all I am going to introduce the term "Machine Learning", this term is the basis of face/object detection of OpenCV.
Why we need to read and understand about Machine Learning? Because of this:
Face detection is a job of DIP (Digital Image Processing) & computer vision; Computer vision is and advance branch of Artificial Intelligence; And Artificial Intelligence is attained via Machine Learning.

In resume, Machine Learning is a way in with help of which implement face detection in computers.

2.4.2.1 Machine Learning

Now let's get deeper about what is Machine Learning:

What is Machine Learning?

The goal of machine learning (ML) is to turn data into information. After learning from a collection of data, we want a machine to be able to answer questions about the data:
What other data is most similar to this data? Is there a house in the image? What ad will the use respond to? There is often a cost component, so this question could become; "Of the products that we make the most money from, which one will the user most likely buy if we show them an ad for it?" Machine learning turns data into information by extracting rules or patterns from that data.

Training and Test Set

Machine learning works on data such as temperature values, stock preices, color intensities, and so on. The data is often preprocessed onto features. We might, for example, take a database of 10,000 face images, run an edge detector on the faces, and the collect features such as edge direction, edge strength, and offset from face center for each face. We might obtain 500 such values per face or a feature vector of 500 entries. We could then use machine learning techniques to construct some kind of model from this collected.

To meet our goals, machine learning algorithms analyze our collected features and adjust weights, thresholds, and other parameters to maximize performance according to those goals. This process of parameter adjustment to meet a goal is what I mean by the term learning.

It is always important to know how well machine learning methods are working, and this can be a subtle task. Traditionally, one breaks up the original data set into a large training set (perhaps 9,000 faces, in my example) and a smaller test set (the remaining 1,000 faces). We can then run our classifier over the training set to learn our age prediction model given the data feature vectors. When we are done, we can test the age prediction classifier on the remaining images in the test set.

2.4.2.2 Face Detection

What is Face Detection? Face detection is a computer vision technology that determines the locations and sizes of human faces in arbitrary (digital) images. Face detection can be regarded as a specific case of object-class detection. In injects-class detection, the task is to find the locations and sizes of all objects in a digital image that belong to a given class.

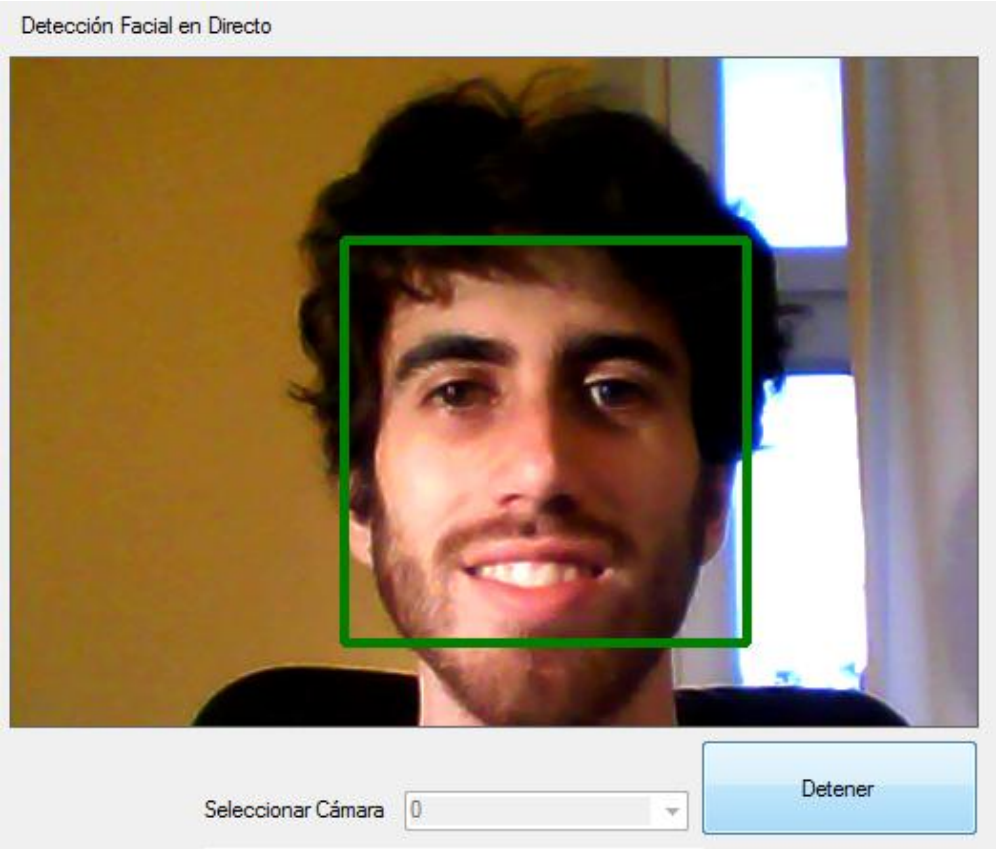

Figure 4 Example of Face Detection

There are many ways to detect a face in a scene, easier and harder ones. Here is a list of the most common approaches on face detection:

- Finding face in images with controlled background.
- Finding faces by color.
- Finding faces by motion.
- Using a mixture of the above.
- Finding faces in unconstrained scenes:
    - Neural Net approach.
    - Neural Nets using statistical cluster information.
    - Model-based Face Tracking.
    - Weak classifier cascades.

Let's get deeper on the problem of Face Detection.

Obviously the most straightforward variety of this problem is the detection of a single face at a known scale and orientation. Even, this, it turns out, is a nontrivial problem. The most immediate application that comes to mind for face detection is as the first step in an automated face recognizer. Thought of this sense, face detection can be applied to systems for things such as automated surveillance and human traffic census.

In and of itself, however, face detection is a fascinating problem; it is an analogue to face tracking that requires no knowledge of previous frames.

Another reason that face detection is an important research problem is its role as a challenging case of more general problem, i.e. object detection, for which the applications, once not restricted to faces, are manifold.

A face is naturally recognizable to a human being despite its many points of variation (e.g. skin tone, hairstyle, facial hair, glasses, etc.). Obviously a human being is able to detect a face in the context of an entire person, but we want a simple, context-free approach to detection. Another source of difficulty for face is the complex 3-dimensional shape of one's face, and the resulting difference in the appearance of given face under different lighting conditions, even in an otherwise identical environment.

The generality of detecting faces in a single grey-scale image is a major challenge.

There are various solutions to this problem, most of which deal with faces at arbitrary (at east, within a reasonable range) scales, though most assume an upright

face. Most of the methods discussed are concerned only with detecting forwards-facing faces.
Scheneiderman and Kanade apply statistical likelihood test, using output histograms to create their detector scheme.

Roowley and Kanade use neural network-based filters in obtaining good early results in what has apparently become a benchmark of sorts for face detection schemes.

Papageorgion et al. propose a general object detection scheme which uses a wavelet representation and statistical learning techniques.

Osuna et al. apply Vapnik's support vector machine technique to face detection.

Romdhani et al. improve on that work by creating reduced training vector sets for their classifier in Fleuret and German attempt a coarse to fine approach to face detection, focusing on minimizing computation in their approach.

But the most impressive paper, **Viola and Jones** use the concept of an "integral image", along with a rectangular feature representation and a boosting algorithm as its learning method, to detect faces at 15 frames per second. This represents an improvement in computation time of an order of magnitude over previous implementations of face detection algorithms.

Mathematical Models and Approaches

Every method wrote here uses a learning algorithm on a training set to begin the detection process. The training stage is extensive for some methods, and relatively small for others.

The most intuitive solution to the problem of modeling faces is the geometric formulation which allows the detector to project a tested image onto a learned subspace and determinate whether or not it is close to that subspace. The natural thing to do with a training se, then, is to compute a manifold in $\mathbb{R}^{\mathbb{N}}$ (from training images containing n pixels) from the most significant components of the face. This is very basic scheme, and is computationally burdensome.

Sung and Poggiouse a "bootstrapping" strategy. A different approach to separating faces and non-faces in image space is used by Osuna et al., and followed up with work by Romdhani et al.

Papageorgiou, Oren, and Poggio, in what can be considered to be a conceptual precursor to the work of **Viola and Jones**, use **Haar** wavelets to create an

overcomplete representation of the face class. The focus of their paper is on the development of their wavelet model.
Following on the work of Papageorgion et al. , viola and jones present a much faster detector than any of their contemporaries. The performance can be attributed to the use of an attentional cascade, using low-feature-number detectors based on a natural extension of **Haar** wavelets. The cascade itself has more to do with their classifier than with their model, so it will be discussed on the next point.

Each detector in their cascade fits objects to simple rectangular masks, basically speaking. In order to avoid making many computations when moving through their cascade, Viola and jones introduce a new image representation which they call an integral image, which is just what is sounds like.

For each pixel in the original image, there is exactly one pixel in the integral image, whose value is the sum of the original image values above and to the left. The integral image can be computed quickly, and drastically improves computation costs under the rectangular feature model.

The integral image allows rectangular sums to be computed in four array references. This is easy to see when the model is considered; under the conventional representation of an image, the computation time needed would b proportional of an image, the computation time needed would be proportional to the size of the rectangle. At the highest levels of the attentional cascade, where most of the comparisons are made, the rectangular features are very large. As the computation progresses down the cascade, the features can get smaller and smaller, but fewer locations are tested for faces. Thus the advantage of the integral image representation is clear. The remaining difficulty lies in creating and training the attentional cascade, which also lends very heavily to the detector's efficiency.

Training the attentional cascade is similar to the other training methods seen, obviously adapted to suit the situation. Because of the cascade's nature, a very high detection rate is needed, but the false detection rate can also be very high, as the overall figures decrease exponentially with the figures for each individual cascade level, based on the depth of the cascade. Each level of the cascade needs to reject example that are closer to faces than the previous level (as each inherits the previous level's accepted images). **Viola and Jones** therefore pass a large number of non-faces examples to train the first cascade level, then pass those detected by the first level on to the next level, and so on. For face training, each level is trained on the same face ser. The method is similar in spirit to the more basic boostrapping methods adapted from Sung's method, but is geared toward the progressive nature of the attentional cascade.

Classifiers

Each model requires a classifier to determine whether given data are faces or non-faces. The classifier is, in general, some threshold applied to the data, usually some sort of goodness of fit measure. The classifiers for the models are discussed now.

Viola and jones use a classifier that, largely for the sake of computational efficiency, is based on an attentional cascade. The individual weak classifiers are based on a variant of the AdaBoost algorithm, which converts weak classifiers into a strong classifier via boosting. To be detected, an image must be detected by each level of a series of basic classifiers, each more discriminating than the last. The computational advantage s gained in the fact that the initial levels of the cascade can use very simple features for their classifiers, and therefore can reject the vast majority of locations in an image quickly. By the time the cascade levels become more meaningful, they are operation only a small proportion of the initial image locations. In their implementation **Viola and Jones** use a 10-layer cascade, each of which contains 20 rectangular features. They compare this against their initial, less efficient detector, which uses 200 rectangular features.
One effect of the cascade strategy is that each classifier must have an extremely high detection rate, but can get away with false positive rates that would in other circumstances be thought abysmal. The reason for this is not hard to see:

The false positive rate F of the entire K-layer cascade is

$$F = \prod_{i=1}^{K} f_{i1}$$

Wheref_i, it is the false positive rate of the ith classifier. Similarly, the cascade's detection rate is

$$D = \prod_{i=1}^{K} d_{i1}$$

So the unusual restraints on detection and rejection rates are obviously justified.

Viola and Jones easily presents the best results in terms of computation time. In terms of their error rates, they provide impressive ROC curves and numerical figures that rival those of Rowley et al.

Viola and Jones seem to have the best application of this principle, and it is aided by the fact that their wavelet representation of their classifiers is flexible enough to

perform meaningful rejection with very little cost. In this sense, the integral image is absolutely the key to a fast application of Haar wavelets.

2.4.2.3 Viola-Jones Method for Face Detection

OpenCV's face detector uses a method that Paul Viola and Michael Jones published in 2001. Usually called simple the Viola-Jones method, or even just Viola-Jones, this approach to detecting objects in images combines four key concepts:

- Simple rectangular features, called Haar features
- An integral Image for rapid feature detection
- The AdaBoost machine-learning method
- A cascaded classifier to combine many features efficiently

The features that Viola and Jones used are based on Haar wavelets. Haar wavelets are single wavelength square waves (one high interval and one low interval). In two dimensions, a square wave is pair of adjacent rectangles, one light and one dark.

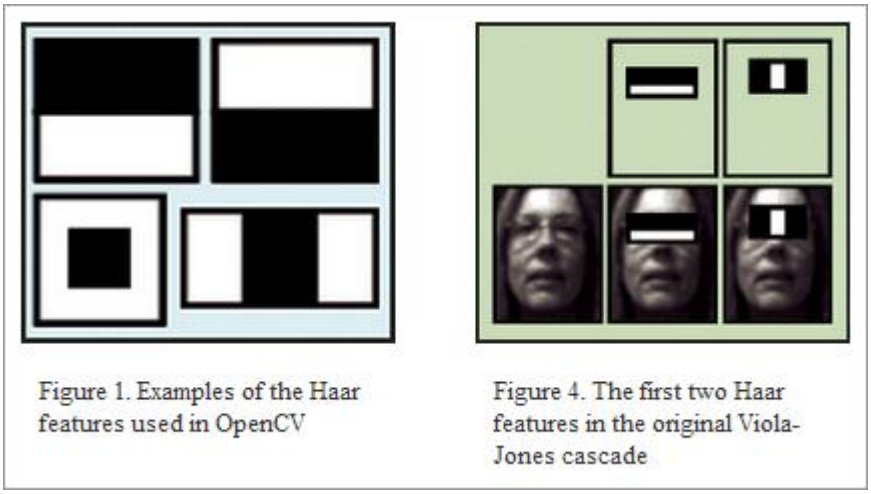

Figure 5 Haar Features used in OpenCV

The actual rectangle combinations used for visual object detection are not true Haar wavelets. Instead, they contain rectangle combinations better suited to visual recognition tasks. Because of that difference, these features are called Haar features, or Haarlike features, rather than Haar wavelets. *Figure 1* shows the features that OpenCV uses.

The presence of a Haar feature is determined by subtracting the average dark-region pixel value from the average light-region pixel value. If the difference is above a threshold (set during learning), that feature is said to be present.

To determine the presence or absence of hundreds of Haar feature at every image location and at several scales efficiently, Viola and Jones used a technique called an Integral Image. In general, "integrating" means adding small units together. In this case, the small units are pixel values. The integral value for each pixel is the sum of all the pixels above it and to its left. Starting at the top left and traversing to the right and down, the entire image can be integrated with a few integer operations per pixel.

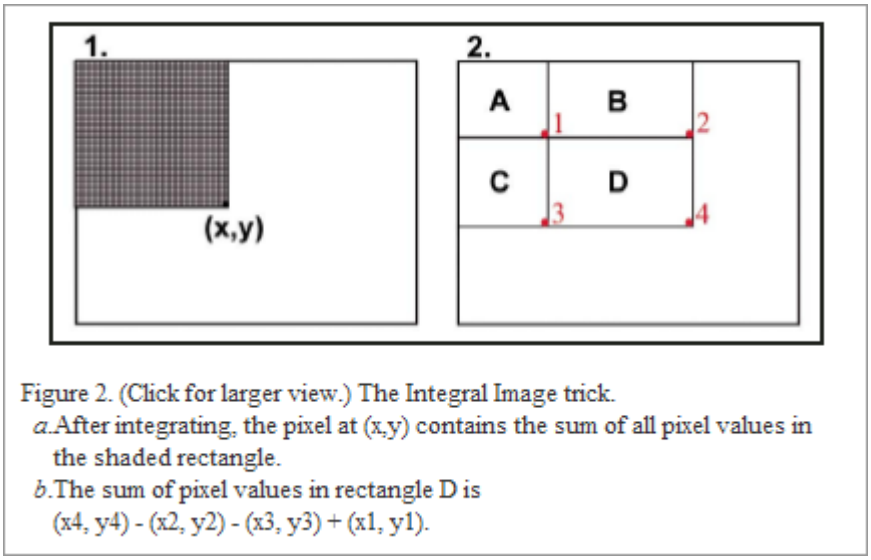

Figure 2. (Click for larger view.) The Integral Image trick.
a.After integrating, the pixel at (x,y) contains the sum of all pixel values in the shaded rectangle.
b.The sum of pixel values in rectangle D is (x4, y4) - (x2, y2) - (x3, y3) + (x1, y1).

Figure 6 Integral Image Trick

As Figure 2.4.2 shows, after integration, the value at each pixel location, (x, y), contains the sum of all pixel values within a rectangular region that has one corner at the top left of the value in this rectangle, you would only need to divide the value at (x, y) by the rectangle's area.

But what if you want to know the summed values for some other rectangle, one that does not have one corner at the upper left of the image? Figure 2.4.2 (Figure 2b) shows the solution to that problem. Suppose you want the summed values in D. You can think of that as being the sum of pixel values in the combined rectangle, A+B+C+D, minus the sums in rectangles A+B and C+D, plus the sum of pixel values in A. In other words,

$$D = A+B+C+D - (A+B) - (A+C) + A$$

Conveniently, A+B+C+D is the Integral Image's value at location 4, A+B is the value at location 2, A+C is the value at location 3, and A is the value at location 1. So, with an Integral Image, you can find the sum of pixel values for any rectangle in the original image with just three integer operations:

$$(x4, y4) – (x2, y2) – (x3, y3) + (x1, y1)$$

To select the specific Haar features to use, and to set threshold levels, Viola and Jones use machine learning method called AdaBoost. Adaboost combines many "weak" classifiers to create one "strong" classifier. "Weak" here means the classifier only gets the right answer a little more often than random guessing would. That is not very good. But if you had a whole lot of there weak classifiers and each one "pushed" the final answer a little bit in the right direction you would have a strong, combine force for arriving at the correct solution. AdaBoost selects a set of weak classifiers to combine and assigns a weight to each this weighted combination is the strong classifier.

Viola and Jones combined a series of AdaBoost classifiers as a filter chain, show in Figure 2.4.3, that is especially efficient for classifying image regions. Each filter is a separate AdaBoost classifier with a fairly small number of weak classifiers.

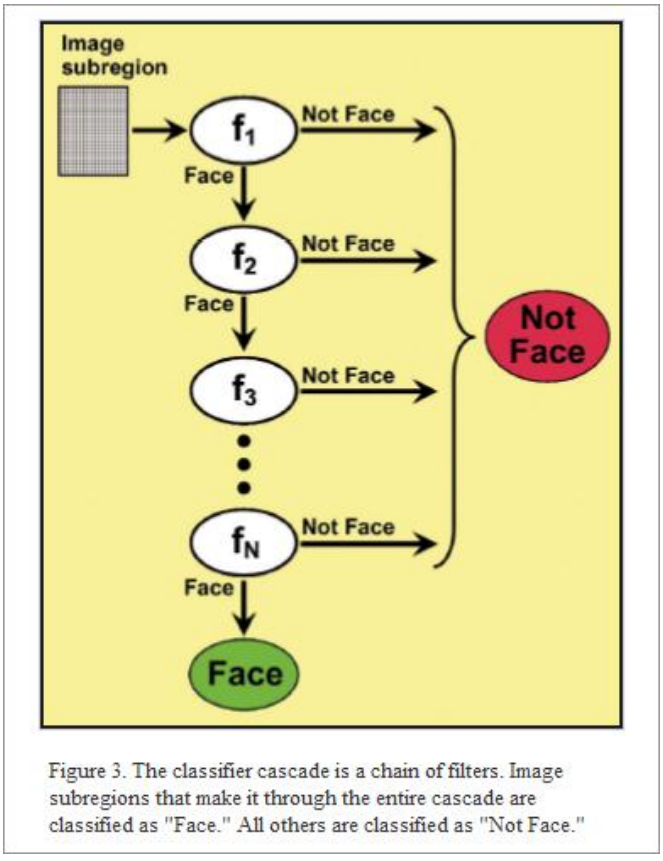

Figure 3. The classifier cascade is a chain of filters. Image subregions that make it through the entire cascade are classified as "Face." All others are classified as "Not Face."

Figure 7 Classifier Cascade

The acceptance threshold at each level is set low enough to pass all, or nearly all, face examples in the training set. The filters at each level are trained to classify training images that passed all previous stages. (the training set is a large database of faces, maybe a thousand or so.) During use, if any one of these filters fails to pass an image region, that region is immediately classified as “not Face”. When a filter passes an image region, it goes to the next filter in the chain. Image region, that pass through all filters in the chain are classified as “Face”. Viola and Jones dubbed this filtering chain a cascade.

The order of filters in the cascade is based on the importance weighting that AdaBoost assigns. The more heavily weighted filters come first, to eliminate non-face image regions as quickly as possible. Figure 2.4.1 (Figure 4) shows the first two features from the original Viola-Jones cascade superimposed on my face. The first one keys off the cheek area being lighter that the eye region. The second uses the fact that the bridge of the nose is lighter than the eyes.

2.4.2.4 Viola And Jones Algorithm and “Haar Cascade Classifier”

This algorithm should be capable of functioning in an unconstrained environment mean in that is should detect all visible faces in any conceivable image. The ideal goal of any face detection algorithm is to perform on par with human inspecting the same image.

The basic problem to be solved is to implement an algorithm for detection of faces in an image. At a first glance the task of face detection may not seem so overwhelming especially considering how easy it is solved by a human. However there is a stark contrast to how difficult it actually is to make a computer successfully solve this task.

In order to ease the task Viola and Jones limit themselves to full view frontal upright faces. That is, in order to be detected the entire face must point towards the camera and it should not be tilted to any side. This may compromise the requirement for being unconstrained a little bit, but considering that the detection algorithm most often will be succeeded by a recognition algorithm these demands seem quite reasonable.

Atypical input image to a face detection algorithm is shown in Figure 2.4.4. This image has a relatively low contrast, contains many different kind of textures and lastly it contains many face. Since more or less all the faces are frontal upright it is the hope that they will be detected by the algorithm developed in this project.

Figure 8 Example of relatively low contrast

Existing methods

During the last decade a number of promising face detection algorithm have been developed and published. Among these three stand out because they are often referred to when performance figures are compared. Here briefly present the outline and main points of each of these algorithms.

Robust Real-Time Objection Detection, 2001
By Paul Viola and Michael J. Jones.
This seems to be the first article where Viola and Jones present the coherent set of ideas that constitute the fundamentals of their face detection algorithm. This algorithm only finds upright faces, but is in 2003 presented in a variant that also detects profile and rotated views.

Neural Network Based Face Detection, 1998
By Henry A. Rowley, Shumeet Baluja and Takeo Kanade.
An image pyramid is calculated in order to detect at multiple scales. A fixed size sub-window is moved through each image in the pyramid. The content of a sub-window is corrected for a non-uniform lightning and subjected to histogram equalization The processed content is fed to several parallel neural networks that carry out the actual face detection. The outputs are combined using logical

AND, thus reducing the amount of false detections. In its first form this algorithm also only detects frontal upright faces.

Statistical Method for 3D Object Detection Applied to Faces and Cars, 2000
By Henry Schneiderman and Takeo Kanade.
The basic mechanics of this algorithm is also to calculate an image pyramid and scan a fixed size sub-window through each layer of this pyramid. The content of the sub-window is subjected to a wavelet analysis and histograms are made for the different wavelet coefficients. These coefficients are fed to differently train parallel detectors that are sensitive to various orientations of the object. The orientation of the object is determined by the detector that yields the highest output. Opposed to the basic Viola-Jones algorithm and the algorithm presented by Rowley et al. this algorithm also detects profile views.

One of the fundamental problems of automated objects detection is that the size and the position of a given object within an image is unknown. As two of the mentioned algorithm demonstrates the standard way to overcome this obstacle is to calculate an image pyramid and scan the detector through each image n the pyramid. While being relatively straightforward to implement this process is rather time consuming, but in their paper Viola-Jones presents a novel approach to this problem.

Methods

The basic principle of the Viola-Jones algorithm is to scan a sub-window capable of detecting faces across a given input image. The standard image processing approach would be to rescale the input image to different sizes and them run he fixed size detector through these images. This approach turns out to be rather time consuming due to the calculation of the different size images.

Contrary to the standard approach Viola-Jones rescale the detector instead of the input image and run the detector many times through the image –each time with a different size images. At first one might suspect both approaches to be equality time consuming, but Viola-Jones have devised a scale invariant detector that requires the same number of calculations whatever the size. This detector is constructed using a so-called integral image and some simple rectangular features reminiscent of Haar wavelets.

The scale invariant detector

The first step of the Viola-Jones face detection algorithm is to turn input image into an integral image. This is done by making each pixel equal to the entire sum of all pixels above to the left of the concerned pixel. Showed if Figure 2.4.5

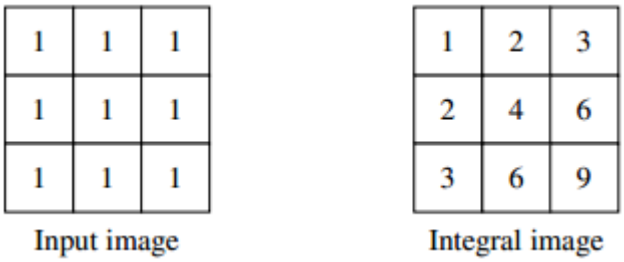

Figure 9 Integral images

This allows for the calculation of the sum of all pixels inside any given rectangle using only four values. These values are the pixels in the integral image that coincide with the corners of the rectangle in the input image. This is demonstrated in Figure 2.4.6

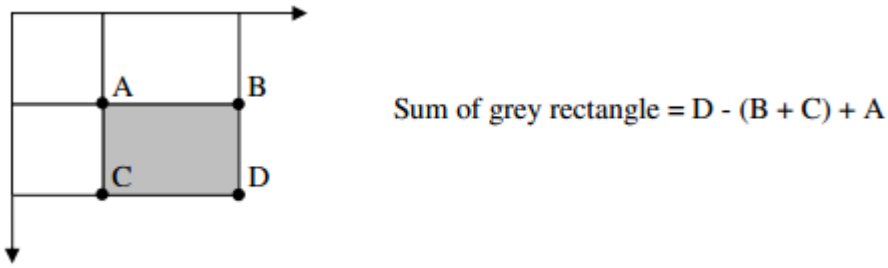

Figure 10 Sum calculation

Since both rectangle B and C include rectangle A the sum of A has to be added to the calculation.

It has now been demonstrated how the sum of pixels within rectangles of arbitrary size can be calculated in constant time. The Viola-Jones face detector analyzes a given sub-window using features consisting of two or more rectangles. The different types of features are shown in Figure 2.4.7

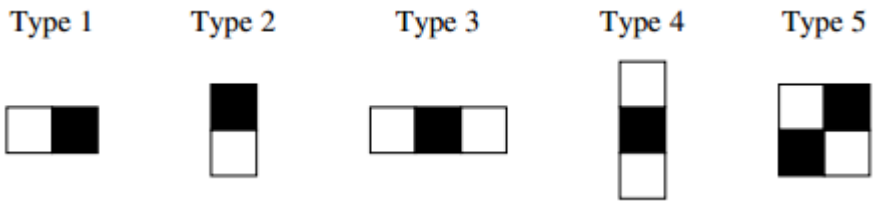

Figure 11 Different types of features

Each feature results in a single value which is calculated by subtracting the sum of the white rectangle(s) from the sum of the black rectangle(s).

Viola-Jones has empirically found that a detector with a base resolution of 24*24 pixels gives satisfactory results. When allowing for all possible sizes and positions of the features in Figure 2.4.7 a total of approximately 160.000 different features can then be constructed. Thus, the amount of possible features vastly outnumbers the 576 pixels contained in the detector at base resolution. These features may seem overly simple to perform such an advanced task as face detection, but what the features lack in complexity they most certainly have in computational efficiency.

One could understand the features as the computer's way of perceiving an input image. The hope being that some features will yield large values when on top of a face. Of course operations could also be carried out directly on the raw pixel, but the variation due to different pose and individual characteristic would be expected to hamper this approach. The goal is now to smartly construct a mesh of features capable of detecting faces and this is the topic of the next section.

The modified AdaBoost algorithm

AdaBoost is a machine learning boosting algorithm capable of constructing a strong classifier through combination of weak classifiers. A weak classifier is mathematically described as:

$$h(x, f, p, \theta) = \begin{cases} 1 & \text{if } pf(x) > p\theta \\ 0 & \text{otherwise} \end{cases}$$

Figure 12 Weak Classifier

Where x is a 24*24 pixel sub-window, f is the applied feature, p the polarity and θ the threshold that decides whether x should be classified as a positive (a face) or a negative (a non-face).

Since only a small amount of the possible 160.000 feature values are expected to be potential weak classifiers the AdaBoost algorithm is modified to select only the best features. Viola-Jones modified AdaBoost algorithm is presented in pseudo code in Figure 2.4.9

- Given examples images $(x_1,y_1),\ldots,(x_n,y_n)$ where y_1=0,1 for negative and positive examples.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for y_1=0,1, where m and l are the numbers of positive and negative examples.
- For t=1,...,T:

1) Normalize the weights, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$

2) Select the best weak classifier with respect to the weighted error:

$$\varepsilon_t = \min_{f,p,\theta} \sum_i w_i \left| h(x_i, f, p, \theta) - y_i \right|$$

3) Define $h_t(x) = h(x, f_t, p_t, \theta_t)$ where f_t, p_t and θ_t are the minimizers of ε_t.

4) Update the weights:

$$w_{t+1,i} = w_{t,i} \beta^{1-e_i}$$

where $e_i = 0$ if example x_i is classified correctly and $e_i = 1$ otherwise, and $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$

- The final strong classifier is:

$$C(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = \log \frac{1}{\beta_t}$

Figure 13 Modified AdaBoost algorithm

An important part of the modified AdaBoost algorithm is the determination of the best feature, polarity and threshold. These seem to be no smart solution to this problem and Viola-Jones suggest a simple brute force method. This means that the determination of each new weak classifier involves evaluation each feature on all the training examples in order to find the best performing feature. This is expected to be the most time consuming part of the training procedure.

The best performing feature is chosen based on the weighted error it procedures. This weighted error is a function of the weights belonging to the training examples, As seen in Figure 2.4.9 part 4) the weight of a correctly classified example is decreased and the weight of a misclassified example is kept constant. As a result it is more “expensive” for the second feature (in the final classifier) to misclassify and example also misclassified by the first feature, than an example classified correctly. An alternative interpretation is that the second feature is forced to focus harder on the examples misclassified by the first. The point being that the weights are a vital part of the mechanics of the AdaBoost algorithm.
With the integral image, the computationally efficient features, and the modified AdaBoost algorithm in pace it seems like the face detector is ready for implementation, but Viola-Jones have one more ace up the sleeve.

The cascaded classifier

The basic principle of the Viola-Jones face detection algorithm is to scan the detector many times through the same image –each time with a new size. Even if an image should contain one or more faces it is obvious that an excessive large amount of the evaluated sub-windows would still be negatives (non-faces). This realization leads to a different formulation of the problem:

Instead of finding faces, the algorithm should discard non-faces.

The thought behind this statement is that it is faster to discard a non-face than to find a face. With this in mind a detector consisting of only one (strong) classifier suddenly seem inefficient since the evaluation time is constant no matter the input. Hence the need for a cascaded classifier arises.

The cascaded classifier is composed of stages each containing a strong classifier. The job of each stage is to determine whether a given sub-window is definitely not a face or maybe a face. When a sub-window is classified to be a non-face by a given stage it is immediately discarded. Conversely a sub-window classified as a maybe face is passed on to the next stage in the cascade. It follows that the more stages a given sub window passes, he higher the chance the sub-window actually contains a face. The concept is illustrated with two stages in Figure 2.4.10

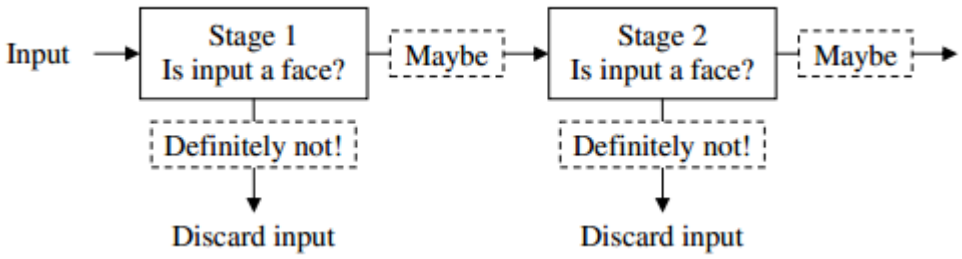

Figure 14 Cascaded Classifier

In a single stage classifier one would normally accept false negatives in order to reduce the false positive rate. However, for the first stage in the staged classifier false positives are not considered to be a problem since the succeeding stages are expected to sort them out. Therefore Viola-Jones prescribes the acceptance of many false positives in the initial stages. Consequently the amount of false negatives in the final staged classifier is expected to be very small.

Viola-Jones also refers to the cascaded classifier as an attentional cascade. This name implies that more attention (computer power) is directed towards the regions of the image suspected to contain faces.

It follows that when training a given stage, say n, the negative examples should of course be false negatives generated by stage n-1.

2.4.3 How to Improve Face Detection

We can make to our own self's, How do I improve the detection results? Or just that Face Detection result is not up to my expectations/needs.

For all of this we can easily solve this problem, we have different options:

1. **"Tune" the face detector to your needs by simple changing the parameter values** in the call to *DetectHaarCascade ()*. I will Explain later.
2. **Try various Haar Cascade XML files avaible for your object (face).**
3. **Ensure good image quality.** The better the quality, the better the detection. Or you could apply image processing for improving image quality.
4. **Make sure the faces in the images are frontal**

2.4.3.1 Improving it Tuning The Detector's Parameters

Now we are going to understand the concepts and use of Haar Cascade classifier's parameter from the EmguCV and C# perspective.

In the main code we can found this function where are all the parameters that we will tune to make our application better, here is:

```
var faces = grayframe.DetectHaarCascade(haar, 1.1, 1,
HAAR_DETECTION_TYPE.DO_CANNY_PRUNING, new Size(ImageFrame.Width / 25,
ImageFrame.Height / 25))[0];
```

1. The Haar Cascade

   The first parameter in the call to DetectHaarCascade () is the XML file rom which, trained data is loaded for the Haar classifier. There are several frontal detector cascades in OpenCV (and EmguCV).

   In my case I use haarcascade frontalface alt tree.xml because it had the best results as I could read in many different places and also I could try, it give me de maximum possible faces.

2. Scale Increase Rate

   The second parameter in the call to DetectHaarCascade() specifies how quickly OpenCV should increase the scale for face detections with each pass it makes over an image. Setting this higher makes the detector run faster, but if it's too high, you may jump too quickly scales and miss faces. The default in OpenCV is 1.1, in other words, scale increases by a factor of 1.1 (10%) each pass.

   This parameter may have a value of 1.1, 1.2, 1.3 or 1.4

3. Minimum Neighbors Threshold

   The third parameter in the call to DetectHaarCascade() is the minimum neighbors threshold which sets the cutoff level for discarding or keeping rectangle groups as "face" or not, based on how many raw detections are in the group. This parameter's value ranges from 0 to 4.

   Isolated detections are false detections, so we should discard these.

   Between isolated rectangles and large grouping, are smaller grouping that may be faces, or may be false detections. The minimum neighbors threshold sets the cutoff level for discarding or keeping rectangle groups based on how many raw detections are in the group.

   For example if I set it to 0, OpenCv will return the complete list of raw detections from the Haar classifier. Let's see an example in the most image use for image processing. Figure 2.4.11

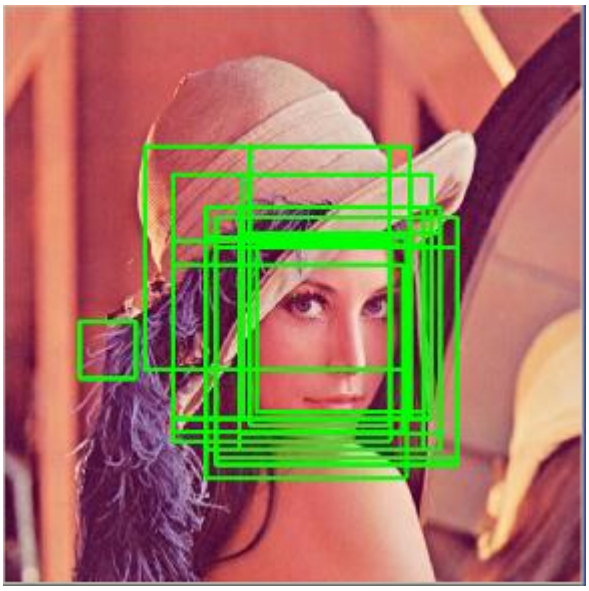

Figure 15 Lena

I have to take into consideration that if you find that your face detector is missing a lot of faces, you might try lowering this threshold to two or one. A small resume:

Higher value → a stricter detection criterion.
Lower value → a lenient detection criterion.

4. Canny Pruning Flag

   The fourth parameter to DetectHaarCascade() is a flag variable. There are currently two options:

   0 or DO_CANNY_PRUNING

   Setting this flag speeds processing, but may cause you miss some faces. Setting the minimum neighbors threshold to 0 so you can view the raw detections will help you better gauge the effect of using Canny pruning.

5. Minimum Detection Scale

   The fifth parameter in the call to DetectHaarCascade() is the size of the smallest face t search for, you can select the default for this by setting the scale to, 25 x 25.

   Depending on the image resolution I am using, this default size may be a very small portion of your overall image. A face image this small may not

meaningful or useful, and detecting it takes up CPU cycles I could use for other purposes. For the reason, it is best to set the minimum detection scale only as small as I truly need.

Chapter 3

Conclusions

The main goal of this project is to create an application were in live Cam the face is detected, later extra parameters were included giving this result. Is a power an stable application with no hard costs, that means that "everyone" could have or create by their self's. In Resume all the objectives have been achieved.

3.1 Final Design of my application

Here is the complete page of my application, on it you could see, Figure 3:

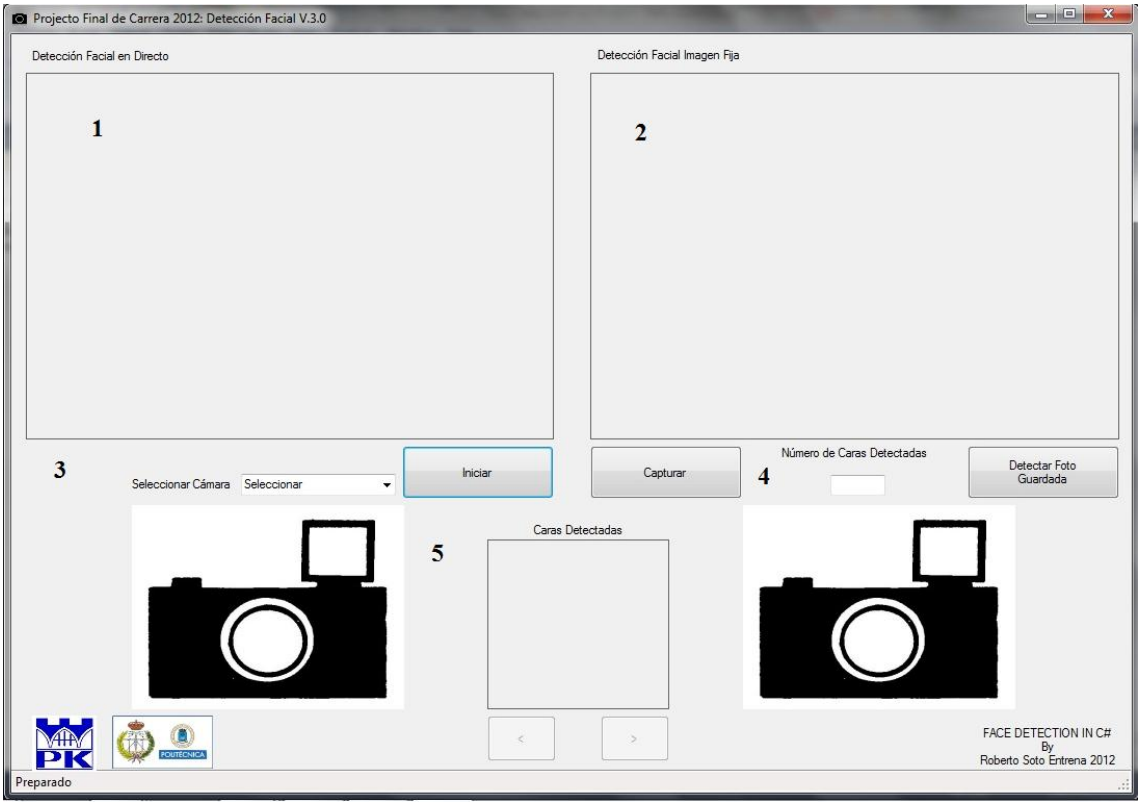

Figure 16 Final Face Detection Application

1. ImageBox, for live Cam Face Detection.
2. PictureBox for Capture Image from live Cam (it can be save), and to charge a load Image and apply Face Detection on it.
3. Identify the Cameras that are connected.
4. Count the number of faces Detected on the Image.

5. Present in the PictureBox just the Face Detected from our image, and in case that there more than one, the user can scroll and see all the Face Detected.

3.2 Examples about how it works

3.2.1 Positive Detections

I will show how my application works with different kinds of extras and later where is so hard for it to detect it.

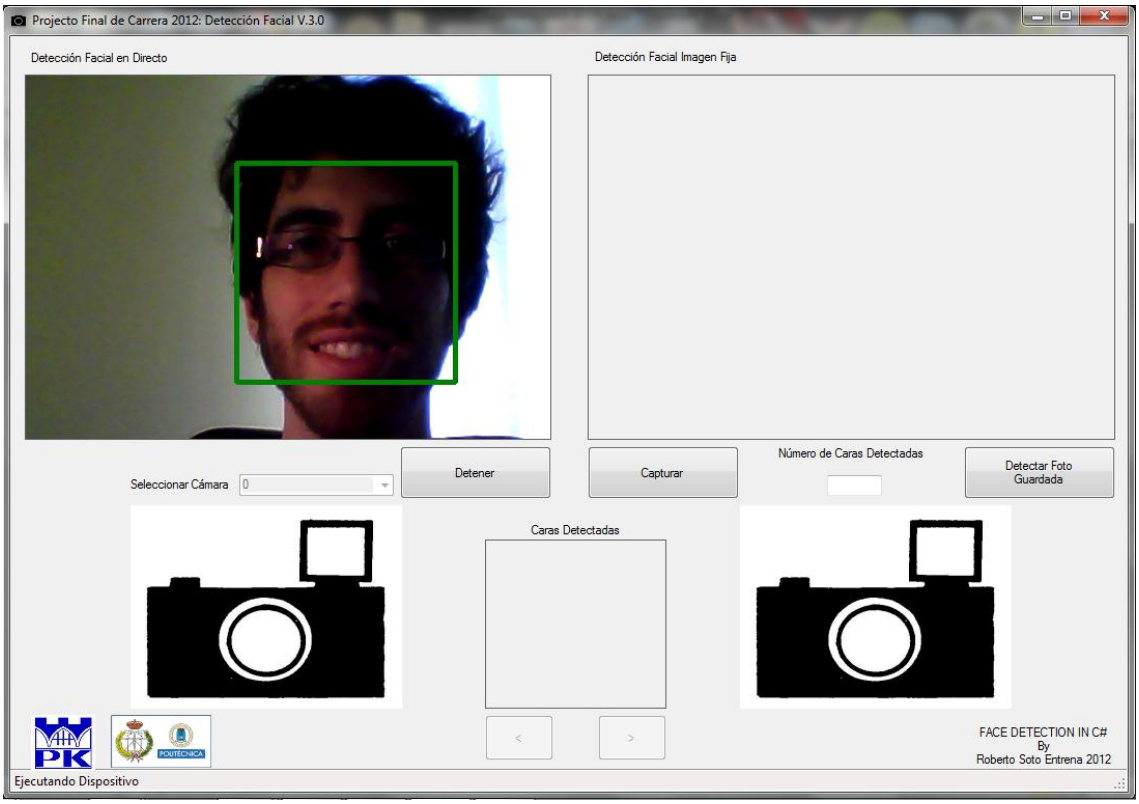

Figure 17 With Glasses

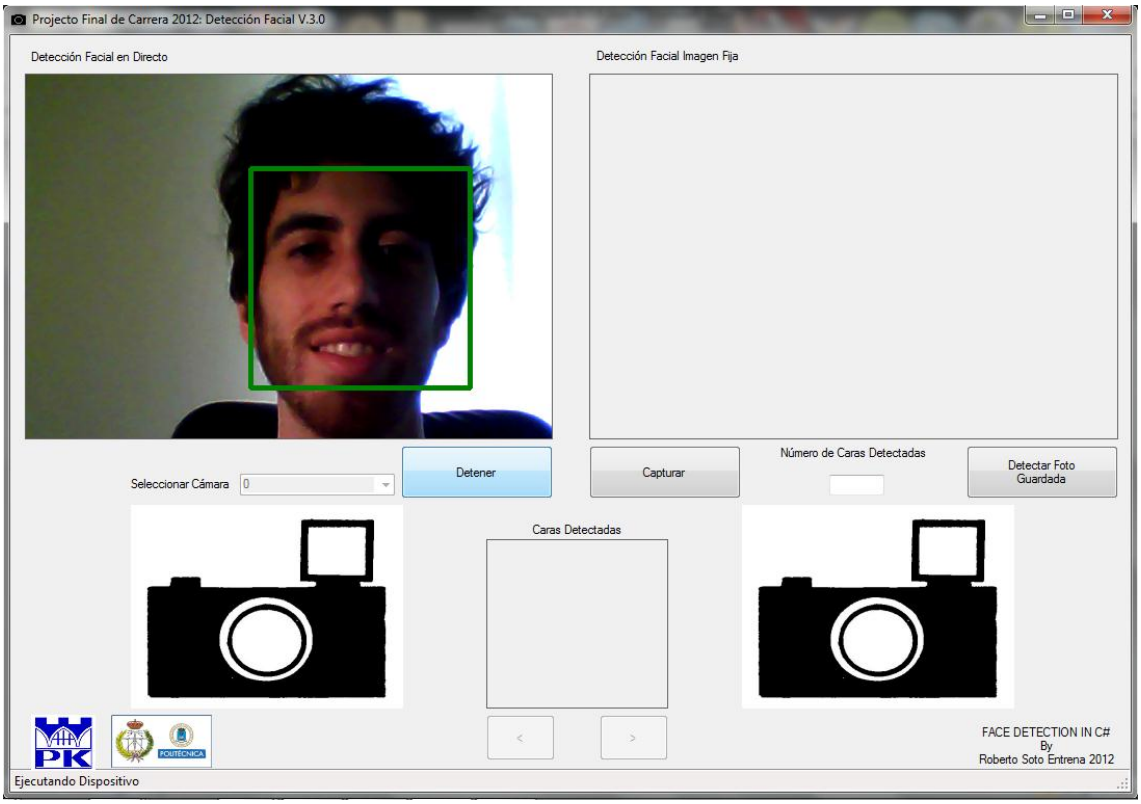

Figure 18 Without Glasses

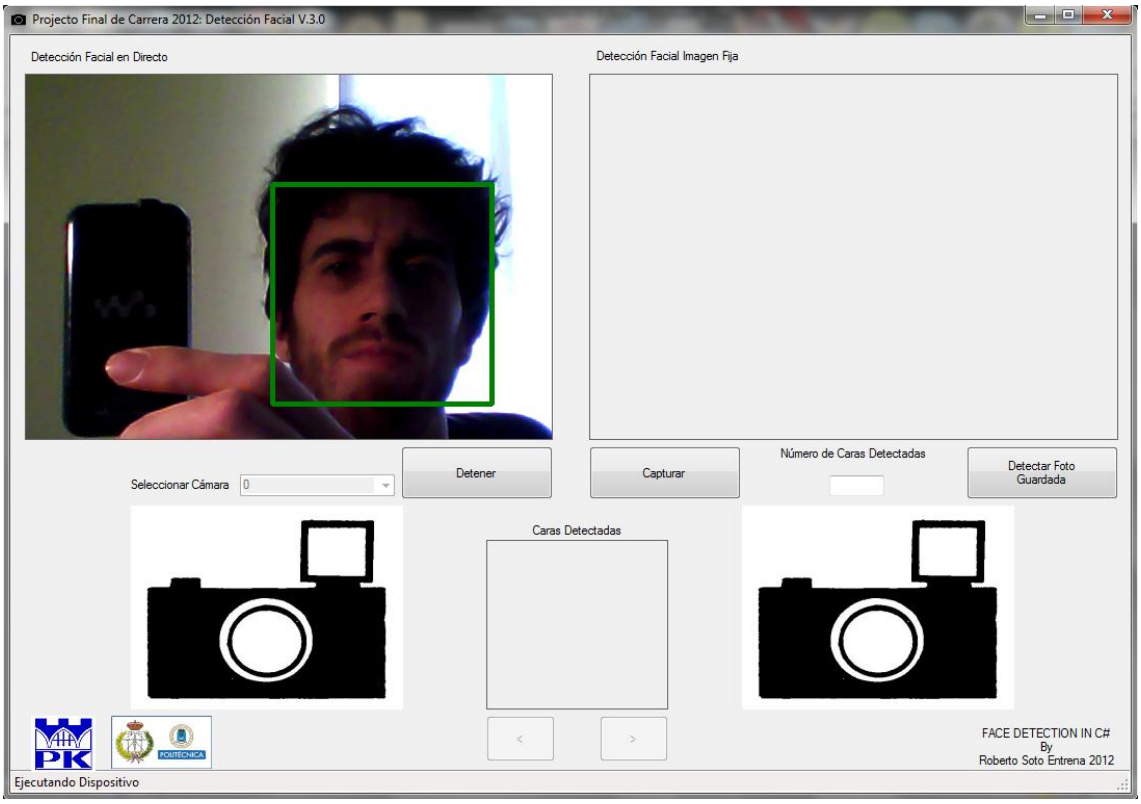

Figure 19 With different objects and different face expression

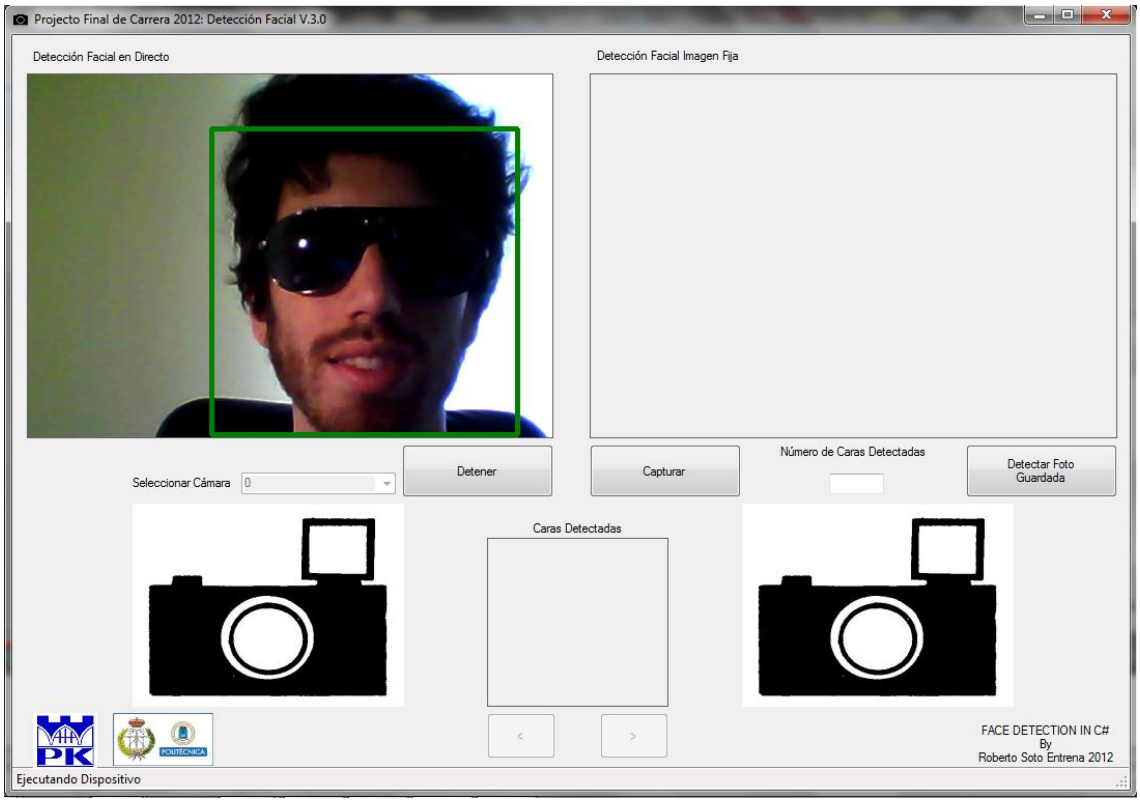

Figure 20 With huge sun glasses

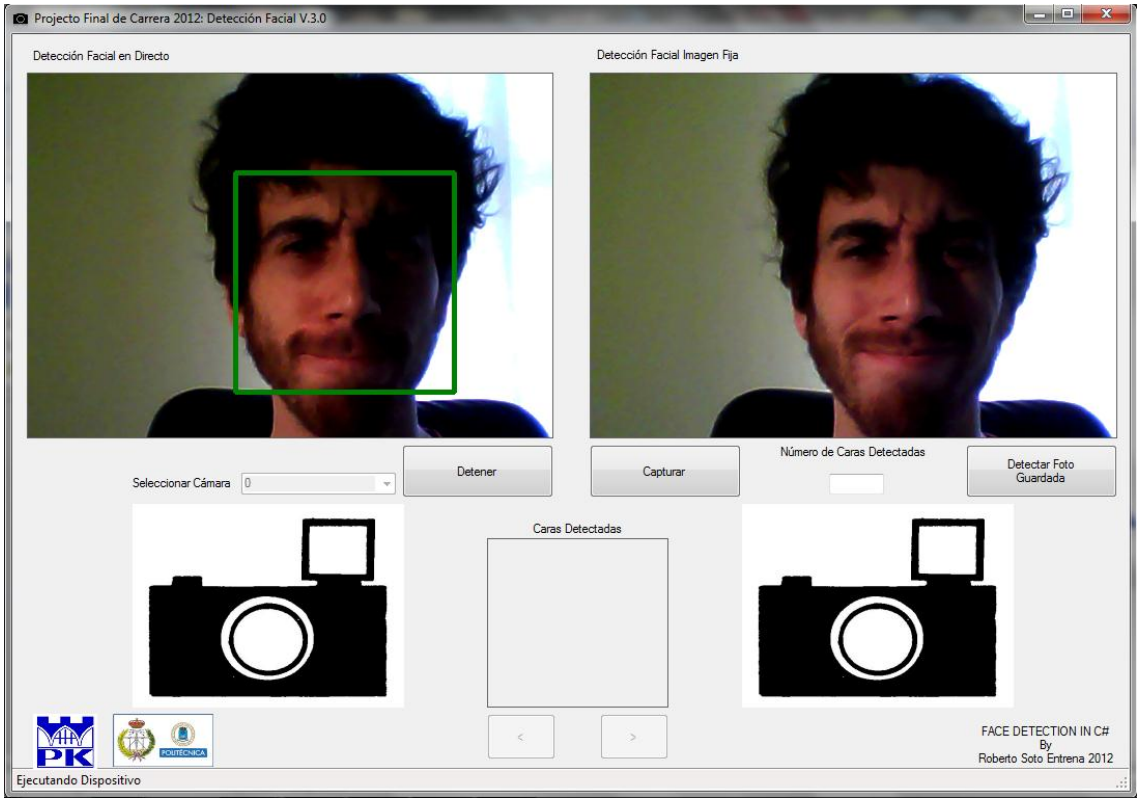

Figure 21 Taking photo from live Face Detection showing it

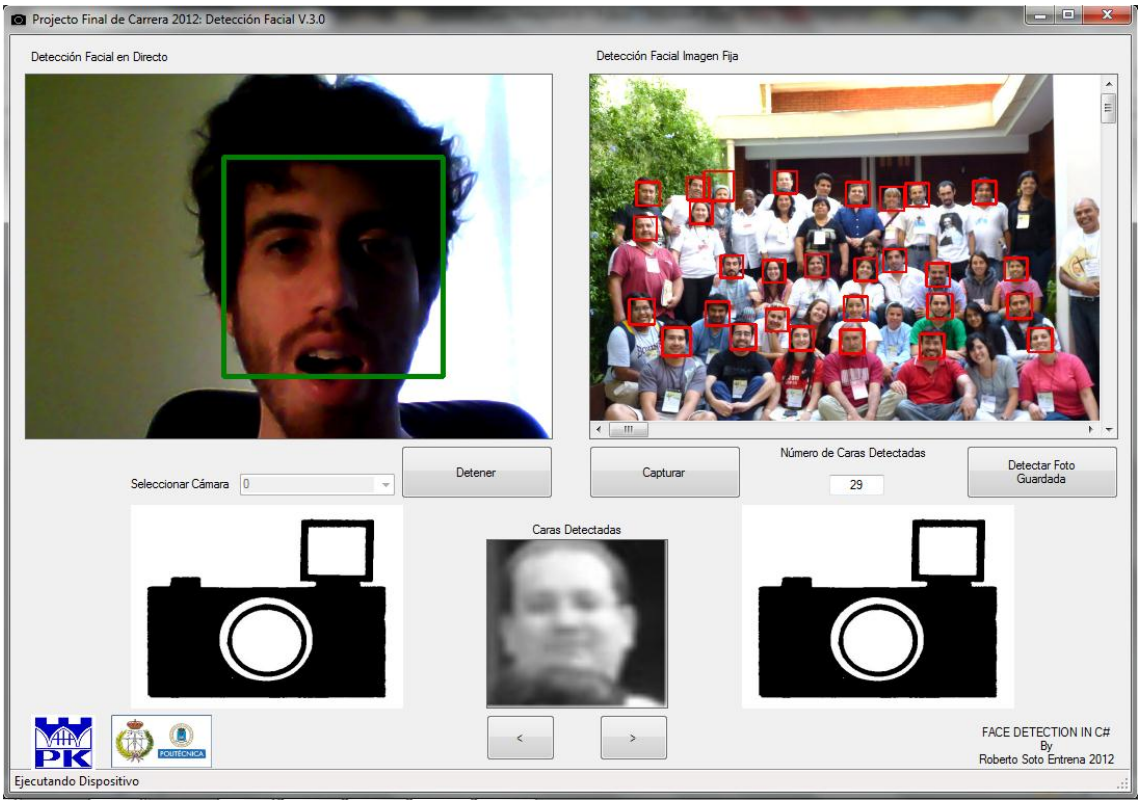

Figure 22 Working Live Detection and load photo with many people, total 29 faces, visble the counter of faces

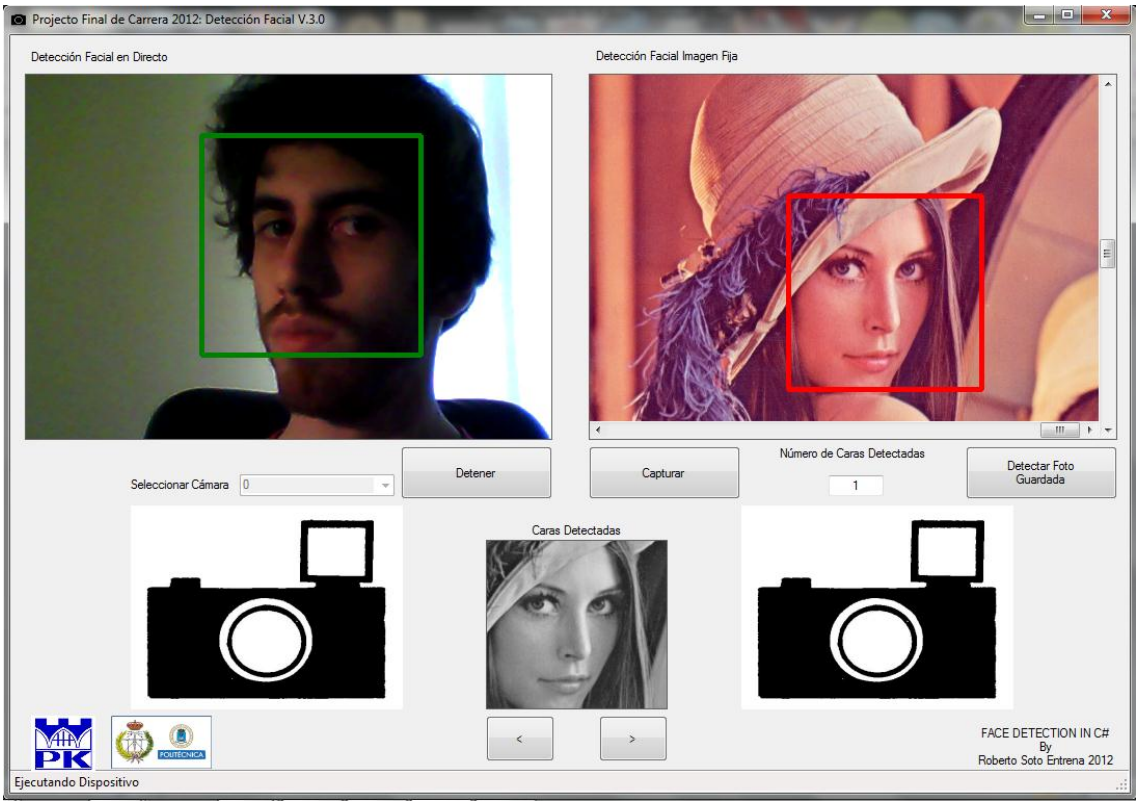

Figure 23 Maximum angles where Face Detection still works and other load image example, Lena

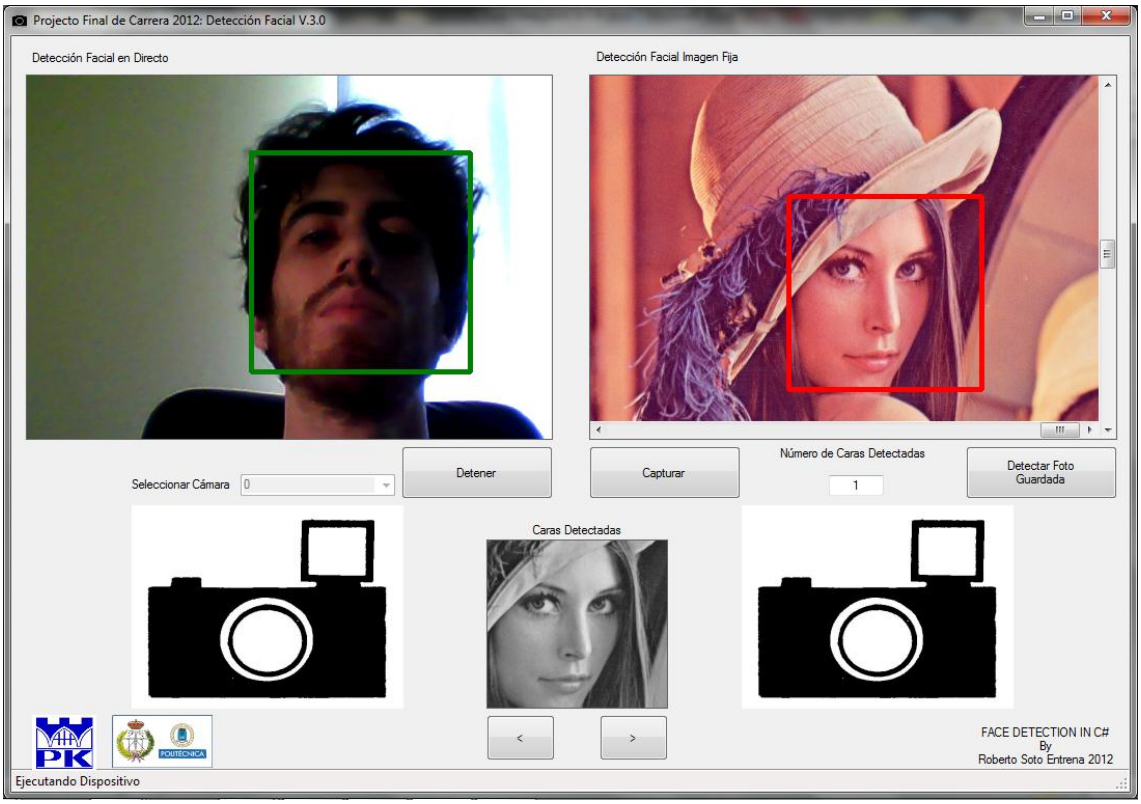

Figure 24 Other example maximum angles Face Detected

3.2.2 Negative Face Detection

Main problem of this algorithm is that have to be frontal Detection, it has maximum inclination of face where it still is Detecting, here I will show it.

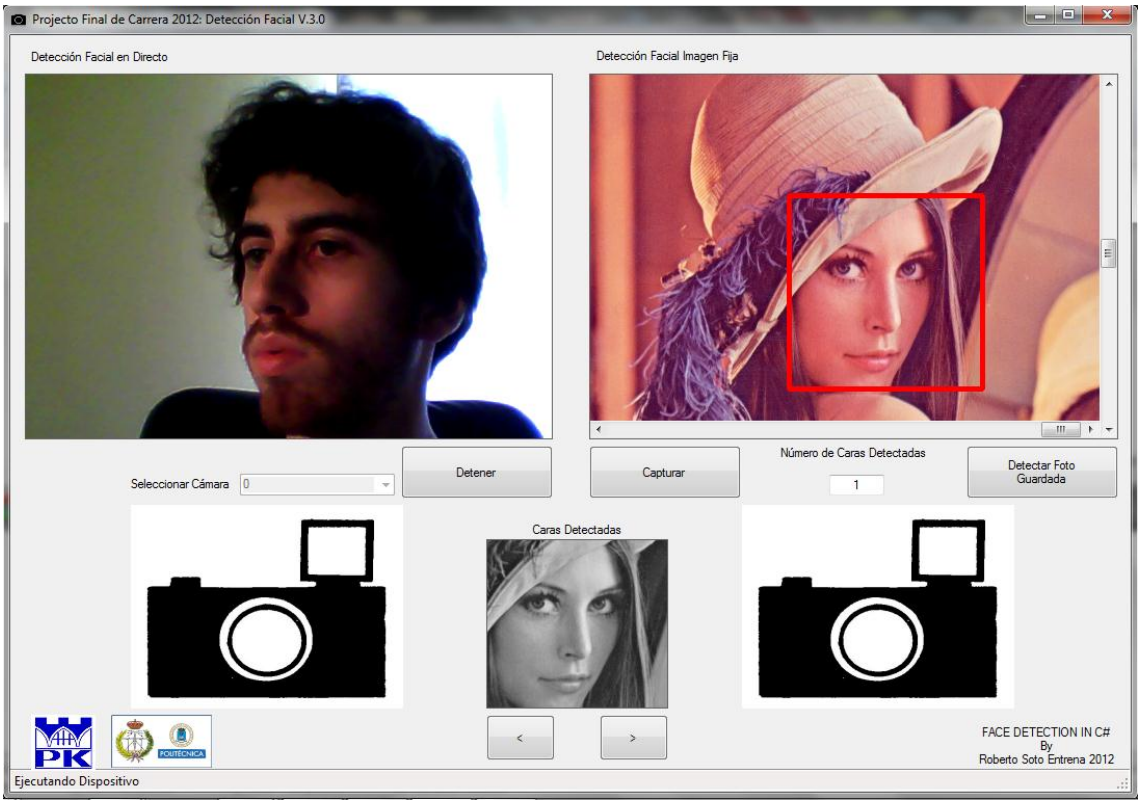

Figure 25 Maximum angles where Face is not Detected

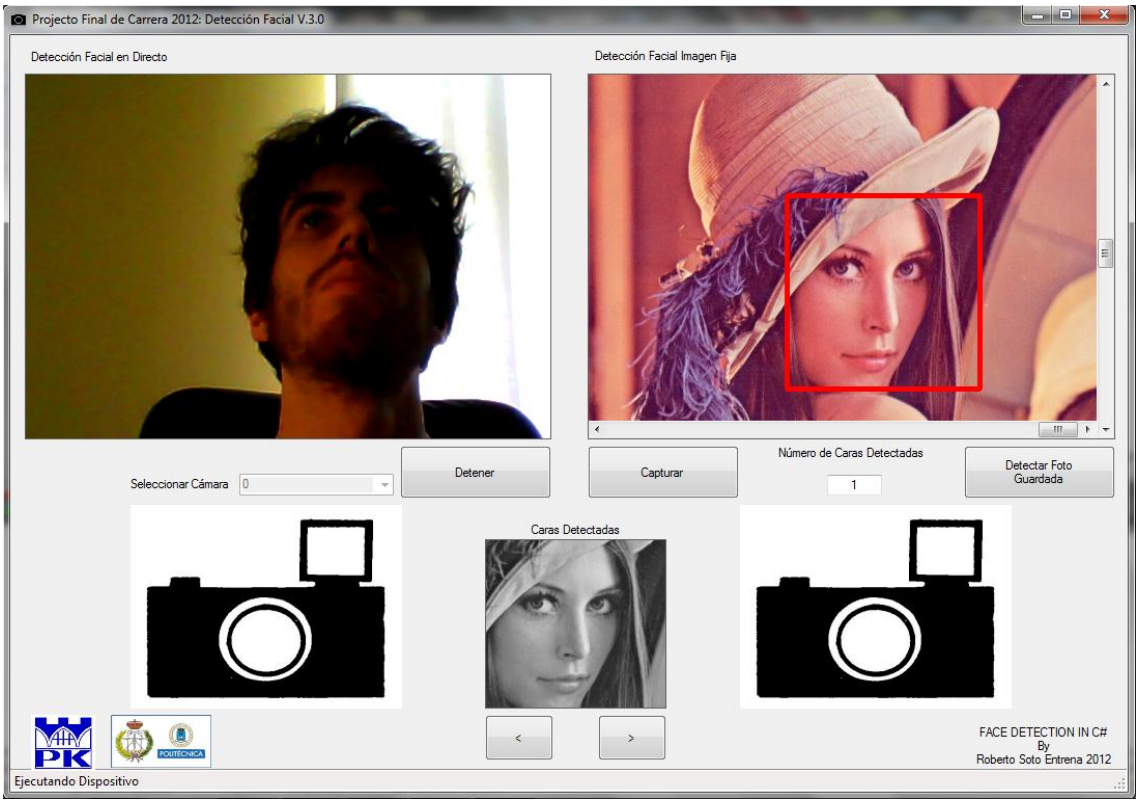

Figure 26 Other example of maximum angle no Detected

Chapter 4

Documentation

4.1 Bibliography

A survey of Methods for Face Detection, A. K. (2003). CiteSeer. Retrieved January 24, 2012, from :

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.119.6322

Cognotics. (n.d.). Cognotics Resources for Cognitive Robotics. Retrieved January 21, 2012, from Seeing with OpenCV:

http://www.cognotics.com/opencv/servo_2007_series/index.html

Frischholz, R. W. (n.d.). Face Detection. Retrieved January 22, 2012, from

http://www.facedetection.com/facedetection/techniques.htm

Jensen, O. H. (n.d.). Implementig the Viola-Jones Face Detecion Algorithm. Retrieved January 20, 2012, from

http://dmpr.cnu.ac.kr/8_seminar/[2012.2]Implementing%20the%20Viola-Jones%20Face%20Detection%20Algorithm.pdf

Jones, P. V. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. Retrieved January 22, 2012, from

http://research.microsoft.com/en-us/um/people/viola/Pubs/Detect/violaJones_CVPR2001.pdf

Jones, P. V. (2003). Robust Real-Time Object Detection. Retrieved January 20, 2012, from

http://research.microsoft.com/en-us/um/people/viola/Pubs/Detect/violaJones_IJCV.pdf

Wiki, E. (n.d.). EmguCV. Retrieved January 20, 2012, from

http://www.emgu.com/wiki/index.php/Main_Page

Wiki, O. (2012, May 08). OpenCV Wiki. Retrieved January 27, 2012, from

http://opencv.willowgarage.com/wiki/

Wikipedia. (n.d.). Viola–Jones object detection framework. Retrieved January 25, 2012, from

http://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework